PDF to Hashcat

```
#!/usr/bin/env python

# Copyright (c) 2013 Shane Quigley, < shane at softwareontheside.info >

# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

# modified to only output hash for hashcat by philsmd, 2015

import re
import sys
import os
from xml.dom import minidom

PY3 = sys.version_info[0] == 3

class PdfParser:
    def __init__(self, file_name):
        self.file_name = file_name
        f = open(file_name, 'rb')
        self.encrypted = f.read()
```

```
        f.close()
        self.process = True
        psr = re.compile(b'PDF-\d\.\d')
        try:
            self.pdf_spec = psr.findall(self.encrypted)[0]
        except IndexError:
            sys.stderr.write("%s is not a PDF file!\n" % file_name)
            self.process = False

    def parse(self):
        if not self.process:
            return

        try:
            trailer = self.get_trailer()
        except RuntimeError:
            e = sys.exc_info()[1]
            sys.stderr.write("%s : %s\n" % (self.file_name, str(e)))
            return
        # print >> sys.stderr, trailer
        object_id = self.get_object_id(b'Encrypt', trailer)
        # print >> sys.stderr, object_id
        if(len(object_id) == 0):
            raise RuntimeError("Could not find object id")
        encryption_dictionary = self.get_encryption_dictionary(object_id)
        # print >> sys.stderr, encryption_dictionary
        dr = re.compile(b'\d+')
        vr = re.compile(b'\/V \d')
        rr = re.compile(b'\/R \d')
        try:
            v = dr.findall(vr.findall(encryption_dictionary)[0])[0]
        except IndexError:
            raise RuntimeError("Could not find /V")
        r = dr.findall(rr.findall(encryption_dictionary)[0])[0]
        lr = re.compile(b'\/Length \d+')
        longest = 0
        # According to the docs:
        # Length : (Optional; PDF 1.4; only if V is 2 or 3). Default value: 40
        length = b'40'
        for le in lr.findall(encryption_dictionary):
            if(int(dr.findall(le)[0]) > longest):
```

```
                longest = int(dr.findall(le)[0])
                length = dr.findall(le)[0]
        pr = re.compile(b'\/P -?\d+')
        try:
            p = pr.findall(encryption_dictionary)[0]
        except IndexError:
                # print >> sys.stderr, "** dict:", encryption_dictionary
                raise RuntimeError("Could not find /P")
        pr = re.compile(b'-?\d+')
        p = pr.findall(p)[0]
        meta = '1' if self.is_meta_data_encrypted(encryption_dictionary) else '0'
        idr = re.compile(b'\/ID\s*\[\s*<\w+>\s*<\w+>\s*\]')
        try:
            i_d = idr.findall(trailer)[0] # id key word
        except IndexError:
            # some pdf files use () instead of <>
            idr = re.compile(b'\/ID\s*\[\s*\(\w+\)\s*\(\w+\)\s*\]')
            try:
                i_d = idr.findall(trailer)[0] # id key word
            except IndexError:
                # print >> sys.stderr, "** idr:", idr
                # print >> sys.stderr, "** trailer:", trailer
                raise RuntimeError("Could not find /ID tag")
                return
        idr = re.compile(b'<\w+>')
        try:
            i_d = idr.findall(trailer)[0]
        except IndexError:
            idr = re.compile(b'\(\w+\)')
            i_d = idr.findall(trailer)[0]
        i_d = i_d.replace(b'<',b'')
        i_d = i_d.replace(b'>',b'')
        i_d = i_d.lower()
        passwords = self.get_passwords_for_JtR(encryption_dictionary)
        output =
'$pdf$'+v.decode('ascii')+'*'+r.decode('ascii')+'*'+length.decode('ascii')+'*'
        output += p.decode('ascii')+'*'+meta+'*'
        output += str(int(len(i_d)/2))+'*'+i_d.decode('ascii')+'*'+passwords
        sys.stdout.write("%s\n" % output.encode('UTF-8'))

    def get_passwords_for_JtR(self, encryption_dictionary):
```

```
    output = ""
    letters = [b"U", b"O"]
    if(b"1.7" in self.pdf_spec):
        letters = [b"U", b"O", b"UE", b"OE"]
    for let in letters:
        pr_str = b'\/' + let + b'\s*\([^)]+\)'
        pr = re.compile(pr_str)
        pas = pr.findall(encryption_dictionary)
        if(len(pas) > 0):
            pas = pr.findall(encryption_dictionary)[0]
            # because regexs in python suck <=== LOL
            while(pas[-2] == b'\\'):
                pr_str += b'[^)]+\)'
                pr = re.compile(pr_str)
                # print >> sys.stderr, "pr_str:", pr_str
                # print >> sys.stderr, encryption_dictionary
                try:
                    pas = pr.findall(encryption_dictionary)[0]
                except IndexError:
                    break
            output += self.get_password_from_byte_string(pas)+"*"
        else:
            pr = re.compile(let + b'\s*<\w+>')
            pas = pr.findall(encryption_dictionary)
            if not pas:
                continue
            pas = pas[0]
            pr = re.compile(b'<\w+>')
            pas = pr.findall(pas)[0]
            pas = pas.replace(b"<",b"")
            pas = pas.replace(b">",b"")
            if PY3:
                output += str(int(len(pas)/2))+'*'+str(pas.lower(),'ascii')+'*'
            else:
                output += str(int(len(pas)/2))+'*'+pas.lower()+'*'
    return output[:-1]

def is_meta_data_encrypted(self, encryption_dictionary):
    mr = re.compile(b'\/EncryptMetadata\s\w+')
    if(len(mr.findall(encryption_dictionary)) > 0):
        wr = re.compile(b'\w+')
```

```
        is_encrypted = wr.findall(mr.findall(encryption_dictionary)[0])[-1]
        if(is_encrypted == b"false"):
            return False
        else:
            return True
    else:
        return True

def parse_meta_data(self, trailer):
    root_object_id = self.get_object_id(b'Root', trailer)
    root_object = self.get_pdf_object(root_object_id)
    object_id = self.get_object_id(b'Metadata', root_object)
    xmp_metadata_object = self.get_pdf_object(object_id)
    return self.get_xmp_values(xmp_metadata_object)

def get_xmp_values(self, xmp_metadata_object):
    xmp_metadata_object = xmp_metadata_object.partition(b"stream")[2]
    xmp_metadata_object = xmp_metadata_object.partition(b"endstream")[0]
    try:
        xml_metadata = minidom.parseString(xmp_metadata_object)
    except:
        return ""
    values = []
    values.append(self.get_dc_value("title", xml_metadata))
    values.append(self.get_dc_value("creator", xml_metadata))
    values.append(self.get_dc_value("description", xml_metadata))
    values.append(self.get_dc_value("subject", xml_metadata))
    created_year = xml_metadata.getElementsByTagName("xmp:CreateDate")
    if(len(created_year) > 0):
        created_year = created_year[0].firstChild.data[0:4]
        values.append(str(created_year))
    return " ".join(values).replace(":", "")

def get_dc_value(self, value, xml_metadata):
    output = xml_metadata.getElementsByTagName("dc:"+value)
    if(len(output) > 0):
        output = output[0]
        output = output.getElementsByTagName("rdf:li")[0]
        if(output.firstChild):
            output = output.firstChild.data
            return output
```

```
        return ""

    def get_encryption_dictionary(self, object_id):
        encryption_dictionary = self.get_pdf_object(object_id)
        for o in encryption_dictionary.split(b"endobj"):
            if(object_id+b" obj" in o):
                encryption_dictionary = o
        return encryption_dictionary

    def get_object_id(self, name , trailer):
        oir = re.compile(b'\/' + name + b'\s\d+\s\d\sR')
        try:
            object_id = oir.findall(trailer)[0]
        except IndexError:
            # print >> sys.stderr, " ** get_object_id: name \"", name, "\", trailer ", trailer
            return ""
        oir = re.compile(b'\d+ \d')
        object_id = oir.findall(object_id)[0]
        return object_id

    def get_pdf_object(self, object_id):
        output = object_id+b" obj" + \
            self.encrypted.partition(b"\r"+object_id+b" obj")[2]
        if(output == object_id+b" obj"):
            output = object_id+b" obj" + \
            self.encrypted.partition(b"\n"+object_id+b" obj")[2]
        output = output.partition(b"endobj")[0] + b"endobj"
        # print >> sys.stderr, output
        return output

    def get_trailer(self):
        trailer = self.get_data_between(b"trailer", b">>", b"/ID")
        if(trailer == b""):
            trailer = self.get_data_between(b"DecodeParms", b"stream", b"")
            if(trailer == ""):
                raise RuntimeError("Can't find trailer")
        if(trailer != "" and trailer.find(b"Encrypt") == -1):
            # print >> sys.stderr, trailer
            raise RuntimeError("File not encrypted")
        return trailer
```

```
def get_data_between(self, s1, s2, tag):
    output = b""
    inside_first = False
    lines = re.split(b'\n|\r', self.encrypted)
    for line in lines:
        inside_first = inside_first or line.find(s1) != -1
        if(inside_first):
            output += line
            if(line.find(s2) != -1):
                if(tag == b"" or output.find(tag) != -1):
                    break
                else:
                    output = b""
                    inside_first = False
    return output

def get_hex_byte(self, o_or_u, i):
    if PY3:
        return hex(o_or_u[i]).replace('0x', '')
    else:
        return hex(ord(o_or_u[i])).replace('0x', '')

def get_password_from_byte_string(self, o_or_u):
    pas = ""
    escape_seq = False
    escapes = 0
    excluded_indexes = [0, 1, 2]
    #For UE & OE in 1.7 spec
    if not PY3:
        if(o_or_u[2] != '('):
            excluded_indexes.append(3)
    else:
        if(o_or_u[2] != 40):
            excluded_indexes.append(3)
    for i in range(len(o_or_u)):
        if(i not in excluded_indexes):
            if(len(self.get_hex_byte(o_or_u, i)) == 1 \
               and o_or_u[i] != "\\"[0]):
                pas += "0"  # need to be 2 digit hex numbers
            is_back_slash = True
            if not PY3:
```

```
                    is_back_slash = o_or_u[i] != "\\"[0]
                else:
                    is_back_slash = o_or_u[i] != 92
                if(is_back_slash or escape_seq):
                    if(escape_seq):
                        if not PY3:
                            esc = "\\"+o_or_u[i]
                        else:
                            esc = "\\"+chr(o_or_u[i])
                        esc = self.unescape(esc)
                        if(len(hex(ord(esc[0])).replace('0x', '')) == 1):
                            pas += "0"
                        pas += hex(ord(esc[0])).replace('0x', '')
                        escape_seq = False
                    else:
                        pas += self.get_hex_byte(o_or_u, i)
                else:
                    escape_seq = True
                    escapes += 1
        output = len(o_or_u)-(len(excluded_indexes)+1)-escapes
        return str(output)+'*'+pas[:-2]

    def unescape(self, esc):
        escape_seq_map = {'\\n':"\n", '\\s':"\s", '\\e':"\e",
                '\\r':"\r", '\\t':"\t", '\\v':"\v", '\\f':"\f",
                '\\b':"\b", '\\a':"\a", "\\)":")",
                "\\(":"(", "\\\\":"\\" }

        return escape_seq_map[esc]

if __name__ == "__main__":
    if len(sys.argv) < 2:
        sys.stderr.write("Usage: %s <PDF file(s)>\n" % \
                         os.path.basename(sys.argv[0]))
        sys.exit(-1)
    for j in range(1, len(sys.argv)):
        if not PY3:
            filename = sys.argv[j].decode('UTF-8')
        else:
            filename = sys.argv[j]
        # sys.stderr.write("Analyzing %s\n" % sys.argv[j].decode('UTF-8'))
```

```
        parser = PdfParser(filename)
        try:
            parser.parse()
        except RuntimeError:
            e = sys.exc_info()[1]
            sys.stderr.write("%s : %s\n" % (filename, str(e)))
```

Revision #1

Created 2025-11-25 18:06:22 UTC by David Rizzo

Updated 2025-11-25 18:06:38 UTC by David Rizzo